Gemini FA timestamps — where frames sit on the clock

Audience & purpose

This notebook is for **anyone** who wants to see **when**, during fluorescein angiography (FA), the APTOS train-set frames fall on the **elapsed FA clock** after OCR. It does **not** classify disease; it **characterizes acquisition timing** so temporal models (and timestamp supervision) can be interpreted honestly.

What is in the file: Each JSON line is one **FA frame**. Important fields:

- **`exam`**: one eye's FA run (e.g. `12_L`).
- **`time_fa_seconds`**: elapsed seconds on the Heidelberg-style FA timer **when the OCR read succeeds**.
- **`status`**: whether that read was accepted (`ok`) or failed.

Here we only use rows where **`status == "ok"`** and `time_fa_seconds` is present (`REQUIRE_OK = True`). Rows without a usable stamp are ignored for the histograms but still counted when listing exams with **no** stamps.

What you will see (reading guide)

1. **Summary counts** — how many frames have stamps, and the min/max FA time across all stamped frames (large max values usually mean **long runs** or **clock/OCR quirks**, not necessarily physiology).
2. **Global histogram** — pooling **all** stamped frames: typical acquisition emphasis is **early FA**; the tail shows exams that continue **many minutes**.
3. **Log-scale histogram** — same data with a **log x-axis** so long-tail structure is visible.
4. **Within-exam normalized position** — each exam's stamped times are scaled to **[0, 1]** from first to last stamp **in that exam**. This removes clock duration across exams and shows **relative sampling density** (where photographers dwell along their own sequence).
5. **Early / Mid / Late coverage** — coarse bins motivated by a clinical schematic (anchors **30 s / 65 s / 330 s**); boundaries are **midpoints between anchors** (see section header). Counts show **how many exams touch each bin at least once**, not frame counts.
6. **HyperF_Type — FA clock span per exam** — using train labels in `config/hyperftype.json`, **duration** is max − min stamped `time_fa_seconds` within each exam (requires ≥2 stamped frames). Box plots and overlapping histograms show how that span differs **by HyperF class** (exploratory; labels describe spatial FA patterns, not protocol).

Caveats (important)

- **OCR errors** shift frames along the time axis; outliers may be **read failures**, not real physiology.
- Phase bins are a **simple operational summary**, not a gold-standard clinical staging rule.
- Comparisons across sites/protocols need care: clocks may **start at injection differently**.
- **HyperF_Type plots** only include exams that appear in **both** the Gemini JSONL and `hyperftype.json` train split, and only if they have **≥2** stamped FA times (otherwise the clock span is undefined). Differences between classes may reflect **confounding** (referral patterns, site), not pathology-driven acquisition length.

Technical (environment)

Plots: Agg backend + PNG embedding via `IPython.display` (avoids `%matplotlib inline` issues on some Matplotlib 3.10+ setups).

Kernel: `./.pixi/envs/default/bin/python` or `pixi run jupyter-register-kernel` → **beyond-last-frame (pixi)**.

Working directory: run from `notebooks/` so `REPO = Path("..").resolve()` finds the repo root.

Data path: `results/gemini_aptos3_train/gemini_aptos3_train.jsonl` (override `JSONL` in the next cell if needed).

```
from __future__ import annotations

import io
import json
from collections import defaultdict
from pathlib import Path

import matplotlib

matplotlib.use("Agg")
import matplotlib.pyplot as plt  # noqa: E402
import numpy as np
from IPython.display import Image, display


def show_fig(fig=None, *, dpi: int = 110) -> None:
    fig = fig or plt.gcf()
    buf = io.BytesIO()
    fig.savefig(buf, format="png", dpi=dpi, bbox_inches="tight")
    buf.seek(0)
    display(Image(data=buf.read()))
    plt.close(fig)


REPO = Path("..").resolve()
```

```python
JSONL = REPO / "results/gemini_aptos3_train/gemini_aptos3_train.jsonl"
REQUIRE_OK = True

rows: list[dict] = []
with JSONL.open(encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        rows.append(json.loads(line))


def fa_seconds(r: dict) -> float | None:
    v = r.get("time_fa_seconds")
    if v is None:
        return None
    if REQUIRE_OK and r.get("status") != "ok":
        return None
    return float(v)


secs = np.array([t for r in rows if (t := fa_seconds(r)) is not None],
dtype=np.float64)
print(f"Rows: {len(rows):,} | Stamped FA seconds
(require_ok={REQUIRE_OK}): {len(secs):,}")
if len(secs):
    print(f"FA clock range: {secs.min():.1f}s – {secs.max():.1f}s
({secs.max()/60:.1f} min max)")
```

```
Rows: 36,048 | Stamped FA seconds (require_ok=True): 32,844
FA clock range: 0.0s – 5398.0s (90.0 min max)
```

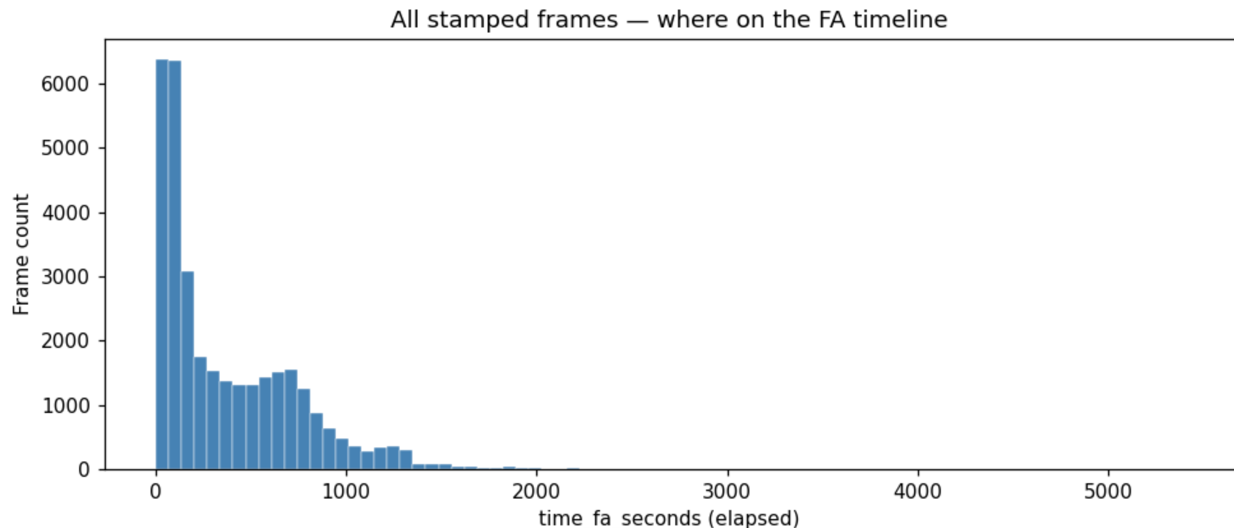
How to read the summary above

- **`Rows`** = total JSON lines (frames) loaded — includes rows **without** a usable FA stamp.
- **`Stamped FA seconds`** = frames contributing to all timeline plots below (`ok` + numeric `time_fa_seconds`). If this is **much smaller** than `Rows`, many frames lack reliable timestamps (limits temporal supervision).
- **`FA clock range`** is across **all stamped frames pooled**. A **large maximum** is normal for datasets that include **late-phase** captures; it does **not** by itself mean every exam reached that time.

The next plots show **where** those stamped seconds concentrate.

```python
# Global distribution (elapsed FA time on the Heidelberg clock)
fig, ax = plt.subplots(figsize=(9, 4))
if len(secs) == 0:
    ax.text(0.5, 0.5, "No stamped rows", ha="center")
else:
    ax.hist(secs, bins=80, color="steelblue", edgecolor="white",
```

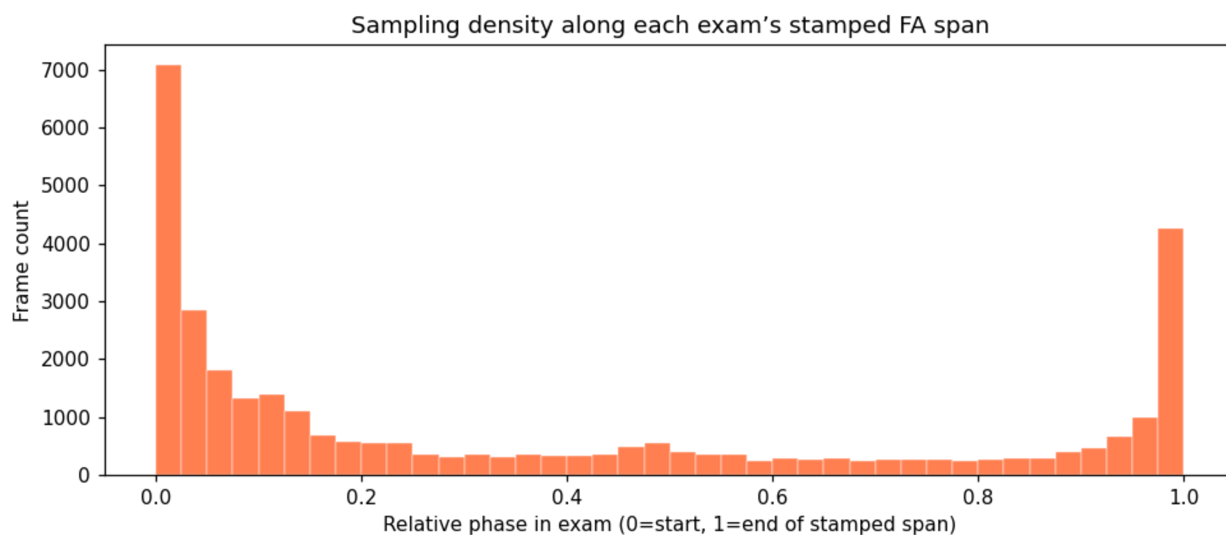
```
linewidth=0.3)
    ax.set_xlabel("time_fa_seconds (elapsed)")
    ax.set_ylabel("Frame count")
    ax.set_title("All stamped frames — where on the FA timeline")
fig.tight_layout()
show_fig(fig)
```



Global histogram — interpretation: The **x-axis** is elapsed FA time in **seconds** for every stamped frame (all exams pooled). A **tall left side** usually means many frames are acquired in **early FA**; the **right tail** shows frames labeled at **later** elapsed times. Shape reflects **both** clinical practice (how long FA runs) and **dataset/OCR behavior** — not a pure biological curve.

```
# Within-exam normalized position: 0 = earliest stamped frame in that
exam, 1 = latest
by_exam: dict[str, list[float]] = defaultdict(list)
for r in rows:
    t = fa_seconds(r)
    if t is None:
        continue
    ex = r.get("exam") or ""
    by_exam[str(ex)].append(t)

norm_fracs: list[float] = []
for ex, ts in by_exam.items():
    if len(ts) < 2:
        continue
    lo, hi = min(ts), max(ts)
    span = hi - lo
    if span <= 0:
        norm_fracs.extend([0.5] * len(ts))
        continue
    norm_fracs.extend((np.array(ts) - lo) / span)
```

```
nf = np.array(norm_fracs, dtype=np.float64)
print(f"Exams with ≥2 stamped frames: {sum(1 for v in by_exam.values()
if len(v) >= 2):,}")
print(f"Contributing frame rows (normalized): {len(nf):,}")

fig, ax = plt.subplots(figsize=(9, 4))
if len(nf):
    ax.hist(nf, bins=40, range=(0, 1), color="coral",
edgecolor="white", linewidth=0.3)
    ax.set_xlabel("Relative phase in exam (0=start, 1=end of stamped
span)")
    ax.set_ylabel("Frame count")
    ax.set_title("Sampling density along each exam's stamped FA span")
fig.tight_layout()
show_fig(fig)

Exams with ≥2 stamped frames: 1,907
Contributing frame rows (normalized): 32,834
```



Within-exam normalized histogram — interpretation: Only exams with **≥2** distinct stamped times contribute (single-time exams cannot define a span). Each frame's FA time is mapped to **0 = earliest** and **1 = latest within that exam**. This removes differences in **total acquisition length** between exams.

- Mass near **0 or 1** → relatively more frames near the **start or end** of that exam's stamped span.
- A **flat-ish** shape → sampling spread more evenly across the span (still depends on frame count and OCR).

If `Contributing frame rows` is much smaller than total stamped rows, many exams have **only one** stamped time or **zero span**.

FA phase coverage per exam (Early / Mid / Late)

Data-driven motivation: Early / mid / late are defined by boundaries that produced the strongest 3-phase separability in embedding space for this dataset.

Operational rule: We use data-driven boundaries found by maximizing 3-phase separability in embedding space:

- **Early:** (t < 103) s
- **Mid:** (103 \le t < 518) s
- **Late:** (t \ge 518) s

Per exam: we only ask whether **at least one** stamped (`status=="ok"`) frame falls in each bin — **presence**, not proportion of frames. Two exams with the same category could still differ greatly in **how many** frames sit in each phase.

Printed table & bar chart: Categories are **mutually exclusive** partitions of `(Early?, Mid?, Late?)` — e.g. **early + late only** means the exam has stamps **both** below 103 s and ≥518 s but **none** in mid. Exams with **no** stamped FA appear in the count line **"Exams with no stamped FA frame"** and are **excluded** from the phase tallies / plot.

How to explain results to others: High counts for **early + mid + late** mean many exams **sample multiple coarse temporal regimes** (under this OCR clock). Dominant **mid + late** or **late only** suggests many sequences **never get an OCR stamp** in the earliest bin — either imaging/OCR starts later, or early frames lack readable timestamps.

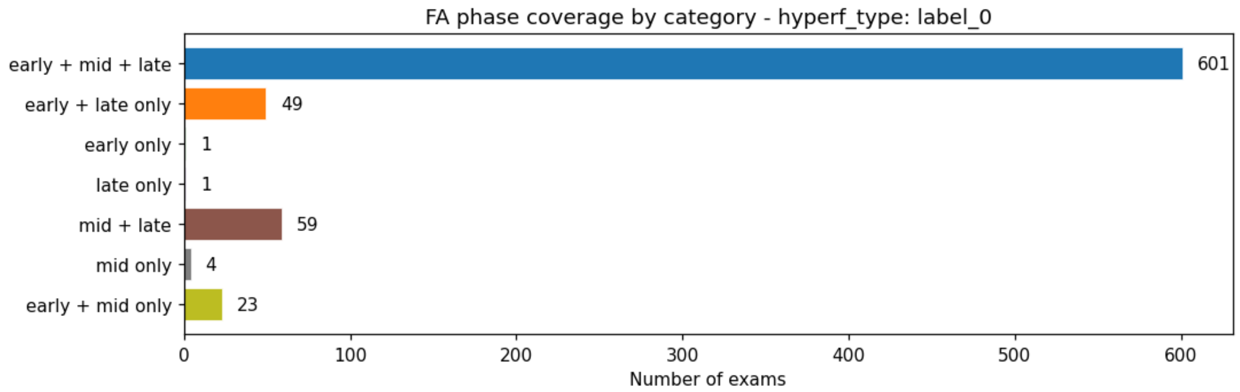
```python
# Corrected: split by HyperF type using exam-level labels from
config/hyperftype.json
import json
from collections import defaultdict


def exam_from_img_path(p: str) -> str:
    # Example: Train/Train/2_L/5.jpg -> 2_L
    parts = str(p).split("/")
    return parts[-2] if len(parts) >= 2 else ""


def hyperf_name(lbl: int) -> str:
    if "LABEL_NAMES" in globals() and isinstance(LABEL_NAMES, dict):
        return LABEL_NAMES.get(lbl, f"label_{lbl}")
    return f"label_{lbl}"


cfg_path = REPO / "config/hyperftype.json"
with cfg_path.open("r", encoding="utf-8") as f:
    cfg = json.load(f)

exam_to_hyperf: dict[str, int] = {}
```

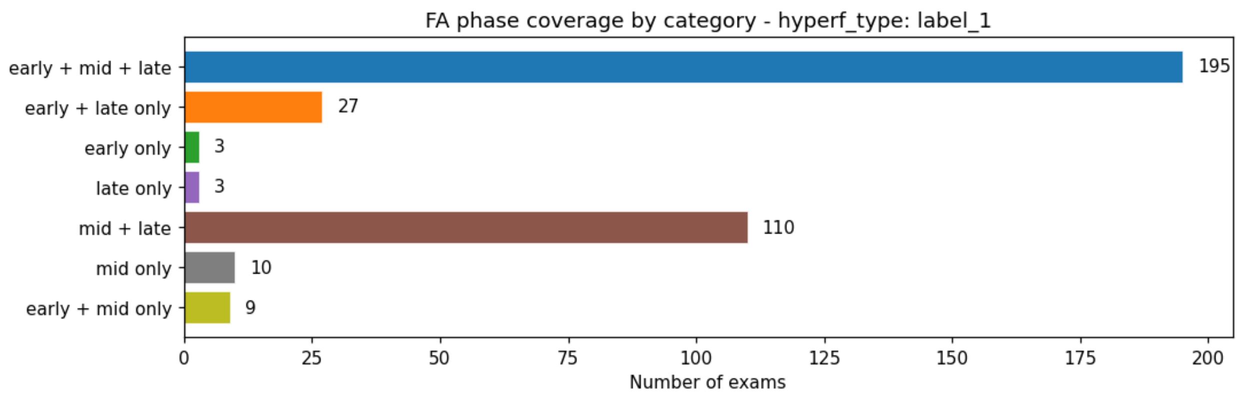
```
for split_name in ("train", "validation", "test"):
    for img_path, lbl in cfg.get(split_name, []):
        ex = exam_from_img_path(img_path)
        if ex and ex not in exam_to_hyperf:
            exam_to_hyperf[ex] = int(lbl)


# Collect stamped FA times per (hyperf_type, exam)
by_type_exam: dict[str, dict[str, list[float]]] = defaultdict(lambda:
defaultdict(list))
missing_label_exams: set[str] = set()

for r in rows:
    t = fa_seconds(r)
    if t is None:
        continue

    ex = str(r.get("exam") or "")
    if not ex:
        continue

    lbl = exam_to_hyperf.get(ex)
    if lbl is None:
        missing_label_exams.add(ex)
        continue

    htype = hyperf_name(lbl)
    by_type_exam[htype][ex].append(t)


# Local fallback definitions (makes this cell self-contained if prior
cells were not run)
if "BOUND_EARLY_MID" not in globals():
    BOUND_EARLY_MID = 103.0
if "BOUND_MID_LATE" not in globals():
    BOUND_MID_LATE = 518.0

if "phase_presence" not in globals():
    def phase_presence(sec: np.ndarray) -> tuple[bool, bool, bool]:
        has_e = bool(np.any(sec < BOUND_EARLY_MID))
        has_m = bool(np.any((sec >= BOUND_EARLY_MID) & (sec <
BOUND_MID_LATE)))
        has_l = bool(np.any(sec >= BOUND_MID_LATE))
        return has_e, has_m, has_l

if "exam_phase_category" not in globals():
    def exam_phase_category(E: bool, M: bool, L: bool) -> str:
        if E and M and L:
            return "early + mid + late"
        if E and not M and L:
```

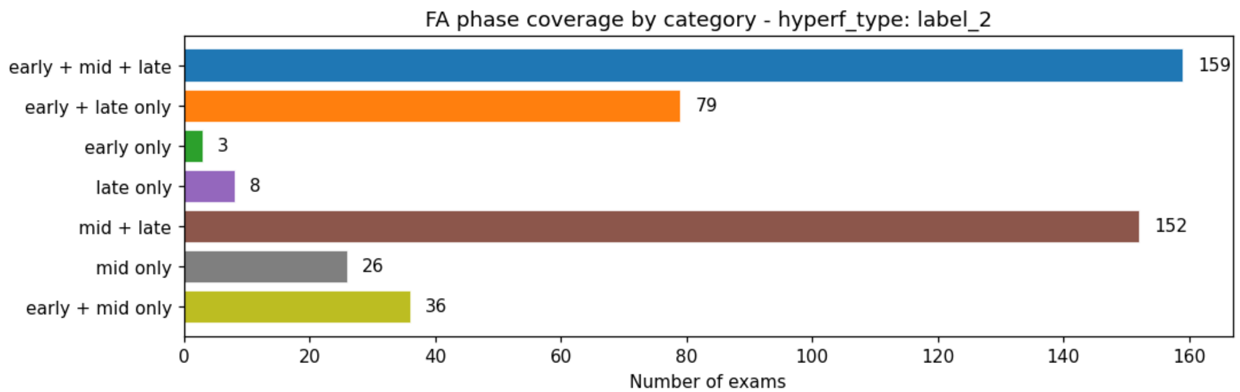
```
            return "early + late only"
        if E and not M and not L:
            return "early only"
        if not E and not M and L:
            return "late only"
        if not E and M and L:
            return "mid + late"
        if not E and M and not L:
            return "mid only"
        if E and M and not L:
            return "early + mid only"
        return "none"

if "order" not in globals():
    order = [
        "early + mid + late",
        "early + late only",
        "early only",
        "late only",
        "mid + late",
        "mid only",
        "early + mid only",
    ]

# Count phase categories per hyperf_type
counts_by_type: dict[str, dict[str, int]] = {}
for htype, exam_map in by_type_exam.items():
    cat_counts_ht = defaultdict(int)
    for ex, ts in exam_map.items():
        ts_arr = np.asarray(ts, dtype=np.float64)
        cat_counts_ht[exam_phase_category(*phase_presence(ts_arr))] +=
1
    counts_by_type[htype] = cat_counts_ht

print(f"Mapped exams with stamped rows: {sum(len(v) for v in
by_type_exam.values()):,}")
if missing_label_exams:
    print(f"Stamped exams without HyperF label:
{len(missing_label_exams):,} (excluded)")


# Plot one chart per hyperf_type
types_sorted = sorted(counts_by_type.keys())
if not types_sorted:
    print("No labeled stamped rows found for per-hyperf_type phase
coverage.")
else:
    fig, axes = plt.subplots(
        nrows=len(types_sorted),
        ncols=1,
```

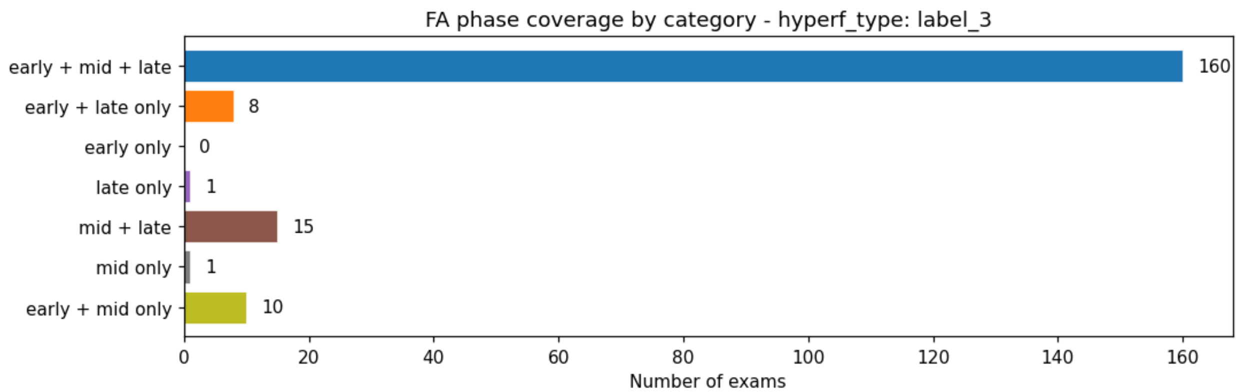
```
        figsize=(10, 3.2 * len(types_sorted)),
        squeeze=False,
    )

    colors = plt.cm.tab10(np.linspace(0, 0.85, len(order)))

    for i, htype in enumerate(types_sorted):
        ax = axes[i, 0]
        cat_counts_ht = counts_by_type[htype]
        counts_plot = [cat_counts_ht[k] for k in order]

        bars = ax.barh(order, counts_plot, color=colors,
edgecolor="white", linewidth=0.5)
        ax.invert_yaxis()
        ax.set_xlabel("Number of exams")
        ax.set_title(f"FA phase coverage by category - hyperf_type:
{htype}")

        xmax = max(counts_plot) if counts_plot else 1
        for b, c in zip(bars, counts_plot):
            ax.text(
                b.get_width() + xmax * 0.015,
                b.get_y() + b.get_height() / 2,
                f"{c:,}",
                va="center",
                fontsize=10,
            )

    fig.tight_layout()
    show_fig(fig)
```

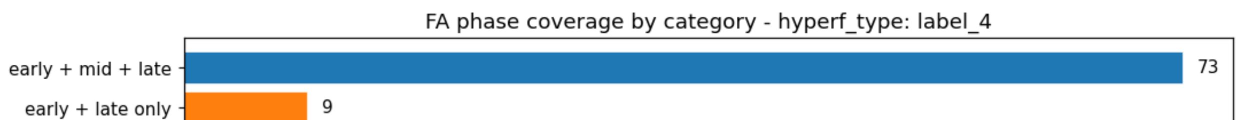
```
Mapped exams with stamped rows: 1,877
Stamped exams without HyperF label: 40 (excluded)
```


FA phase coverage by category - hyperf_type: label_0
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
601
49
1
1
59
4
23
0
100
200
300
400
500
600
Number of exams


FA phase coverage by category - hyperf_type: label_1
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
195
27
3
3
110
10
9
0
25
50
75
100
125
150
175
200
Number of exams


FA phase coverage by category - hyperf_type: label_2
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
159
79
3
8
152
26
36
0
20
40
60
80
100
120
140
160
Number of exams


FA phase coverage by category - hyperf_type: label_3
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
160
8
0
1
15
1
10
0
20
40
60
80
100
120
140
160
Number of exams

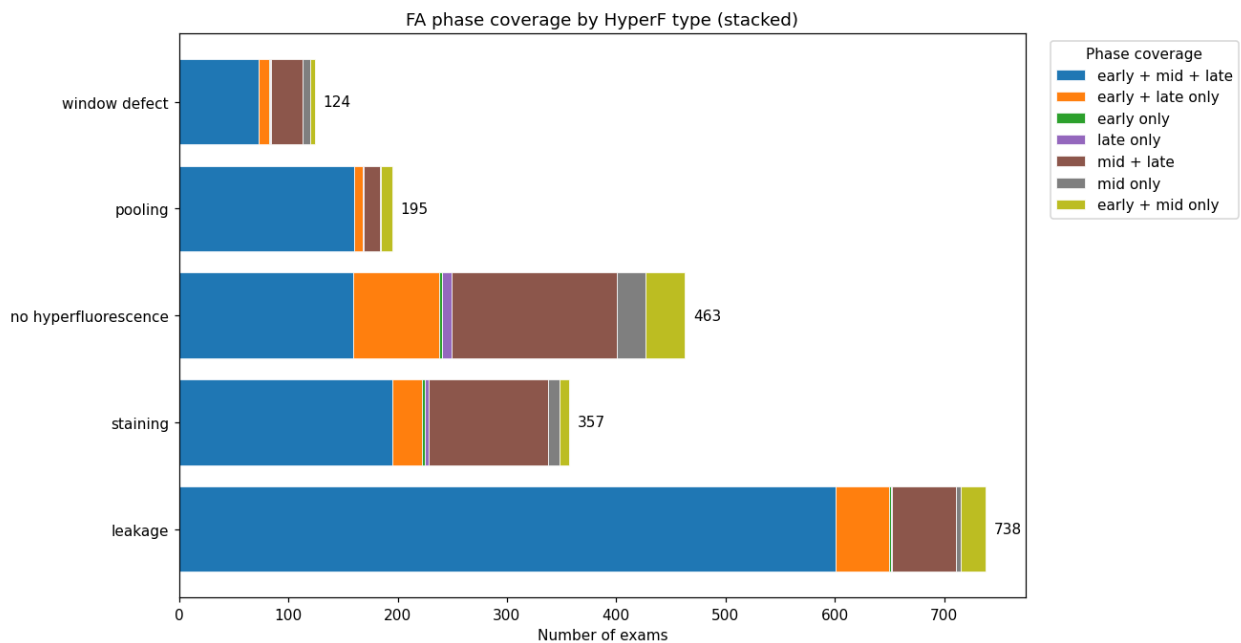

FA phase coverage by category - hyperf_type: label_4
early + mid + late
early + late only
73
9

```python
# Same phase-coverage breakdown, split by HyperF type (named) in one
stacked chart
import json
from collections import defaultdict


def exam_from_img_path(p: str) -> str:
    # Example: Train/Train/2_L/5.jpg -> 2_L
    parts = str(p).split("/")
    return parts[-2] if len(parts) >= 2 else ""


# Canonical HyperF labels used in this repo (see
scripts/probing/utils.py)
HYPERF_LABEL_NAMES = {
    0: "leakage",
    1: "staining",
    2: "no hyperfluorescence",
    3: "pooling",
    4: "window defect",
}


cfg_path = REPO / "config/hyperftype.json"
with cfg_path.open("r", encoding="utf-8") as f:
    cfg = json.load(f)

exam_to_hyperf: dict[str, int] = {}
for split_name in ("train", "validation", "test"):
    for img_path, lbl in cfg.get(split_name, []):
        ex = exam_from_img_path(img_path)
        if ex and ex not in exam_to_hyperf:
            exam_to_hyperf[ex] = int(lbl)


# Collect stamped FA times per (label_id, exam)
by_label_exam: dict[int, dict[str, list[float]]] = defaultdict(lambda:
defaultdict(list))
missing_label_exams: set[str] = set()

for r in rows:
    t = fa_seconds(r)
    if t is None:
        continue

    ex = str(r.get("exam") or "")
    if not ex:
        continue

    lbl = exam_to_hyperf.get(ex)
    if lbl is None:
```

```
        missing_label_exams.add(ex)
        continue

    by_label_exam[int(lbl)][ex].append(t)


# Count phase categories per HyperF label
counts_by_label: dict[int, dict[str, int]] = {}
for lbl, exam_map in by_label_exam.items():
    cat_counts = defaultdict(int)
    for ex, ts in exam_map.items():
        ts_arr = np.asarray(ts, dtype=np.float64)
        cat_counts[exam_phase_category(*phase_presence(ts_arr))] += 1
    counts_by_label[lbl] = cat_counts

print(f"Mapped exams with stamped rows: {sum(len(v) for v in
by_label_exam.values()):,}")
if missing_label_exams:
    print(f"Stamped exams without HyperF label:
{len(missing_label_exams):,} (excluded)")


labels_sorted = sorted(counts_by_label.keys())
if not labels_sorted:
    print("No labeled stamped rows found for per-hyperf_type phase
coverage.")
else:
    type_names = [HYPERF_LABEL_NAMES.get(lbl, f"label_{lbl}") for lbl
in labels_sorted]

    # rows: hyperf type, cols: phase coverage category
    data = np.array(
        [[counts_by_label[lbl].get(cat, 0) for cat in order] for lbl
in labels_sorted],
        dtype=np.int64,
    )

    fig, ax = plt.subplots(figsize=(12, max(4.0, 0.9 * len(type_names)
+ 1.8)))
    colors = plt.cm.tab10(np.linspace(0, 0.85, len(order)))

    left = np.zeros(len(type_names), dtype=np.int64)
    for j, cat in enumerate(order):
        vals = data[:, j]
        ax.barh(type_names, vals, left=left, color=colors[j],
edgecolor="white", linewidth=0.5, label=cat)
        left += vals

    # Show totals at the end of each stacked bar
    for i, total in enumerate(left.tolist()):
        ax.text(total + max(1, left.max() * 0.01), i, f"{total:,}",
```

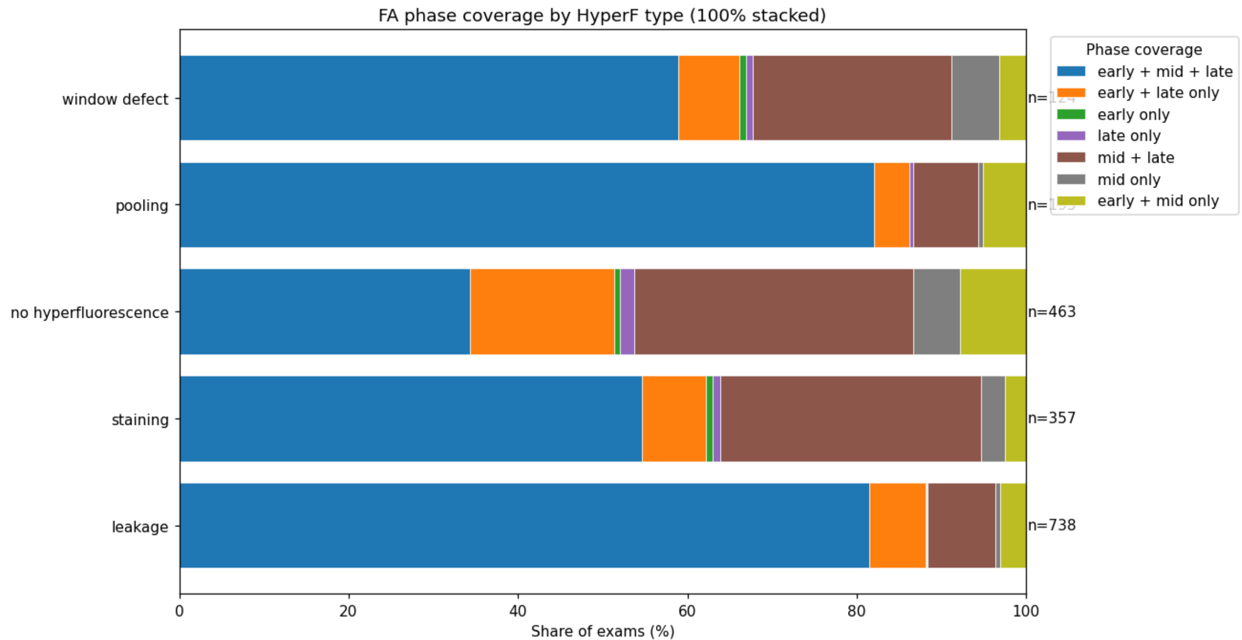
```
va="center", fontsize=10)

    ax.set_xlabel("Number of exams")
    ax.set_title("FA phase coverage by HyperF type (stacked)")
    ax.legend(title="Phase coverage", bbox_to_anchor=(1.02, 1),
loc="upper left")
    fig.tight_layout()
    show_fig(fig)
```

```
Mapped exams with stamped rows: 1,877
Stamped exams without HyperF label: 40 (excluded)
```




```
# Percentage view: FA phase coverage by HyperF type (100% stacked)
# Reuses: counts_by_label, labels_sorted, order, HYPERF_LABEL_NAMES

if not labels_sorted:
    print("No labeled stamped rows found for percentage plot.")
else:
    type_names = [HYPERF_LABEL_NAMES.get(lbl, f"label_{lbl}") for lbl
in labels_sorted]

    count_data = np.array(
        [[counts_by_label[lbl].get(cat, 0) for cat in order] for lbl
in labels_sorted],
        dtype=np.float64,
    )
    totals = count_data.sum(axis=1, keepdims=True)

    # Avoid divide-by-zero for rare empty rows
    pct_data = np.divide(count_data, totals,
```

```
out=np.zeros_like(count_data), where=totals > 0) * 100.0

    fig, ax = plt.subplots(figsize=(12, max(4.0, 0.9 * len(type_names)
+ 1.8)))
    colors = plt.cm.tab10(np.linspace(0, 0.85, len(order)))

    left = np.zeros(len(type_names), dtype=np.float64)
    for j, cat in enumerate(order):
        vals = pct_data[:, j]
        ax.barh(
            type_names,
            vals,
            left=left,
            color=colors[j],
            edgecolor="white",
            linewidth=0.5,
            label=cat,
        )
        left += vals

    # 100% reference and sample sizes
    ax.set_xlim(0, 100)
    for i, n in enumerate(totals[:, 0].astype(int).tolist()):
        ax.text(100.2, i, f"n={n:,}", va="center", fontsize=10)

    ax.set_xlabel("Share of exams (%)")
    ax.set_title("FA phase coverage by HyperF type (100% stacked)")
    ax.legend(title="Phase coverage", bbox_to_anchor=(1.02, 1),
loc="upper left")
    fig.tight_layout()
    show_fig(fig)
```

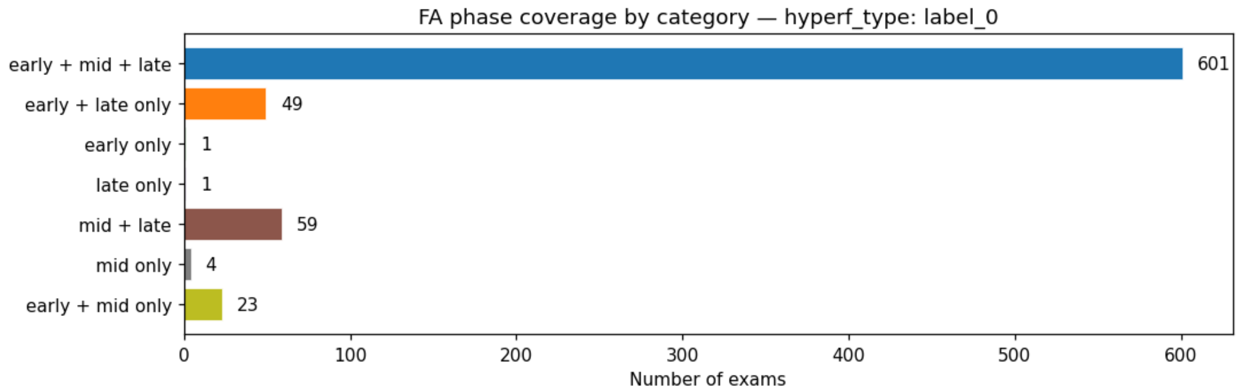


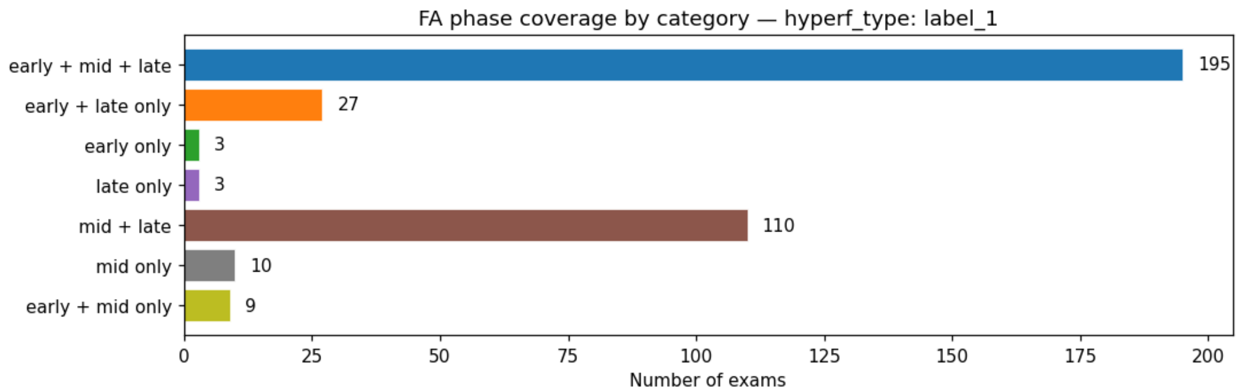

```
# Corrected: split by HyperF type using exam-level labels from
config/hyperftype.json
import json

def exam_from_img_path(p: str) -> str:
    # Example: Train/Train/2_L/5.jpg -> 2_L
    parts = str(p).split("/")
    return parts[-2] if len(parts) >= 2 else ""


def hyperf_name(lbl: int) -> str:
    # Reuse notebook label names if already defined; otherwise
readable fallback.
    if "LABEL_NAMES" in globals() and isinstance(LABEL_NAMES, dict):
        return LABEL_NAMES.get(lbl, f"label_{lbl}")
    return f"label_{lbl}"


# Build exam -> hyperf_type mapping
cfg_path = REPO / "config/hyperftype.json"
with cfg_path.open("r", encoding="utf-8") as f:
    cfg = json.load(f)

exam_to_hyperf: dict[str, int] = {}
for split_name in ("train", "validation", "test"):
    for img_path, lbl in cfg.get(split_name, []):
        ex = exam_from_img_path(img_path)
        if ex and ex not in exam_to_hyperf:
            exam_to_hyperf[ex] = int(lbl)
```

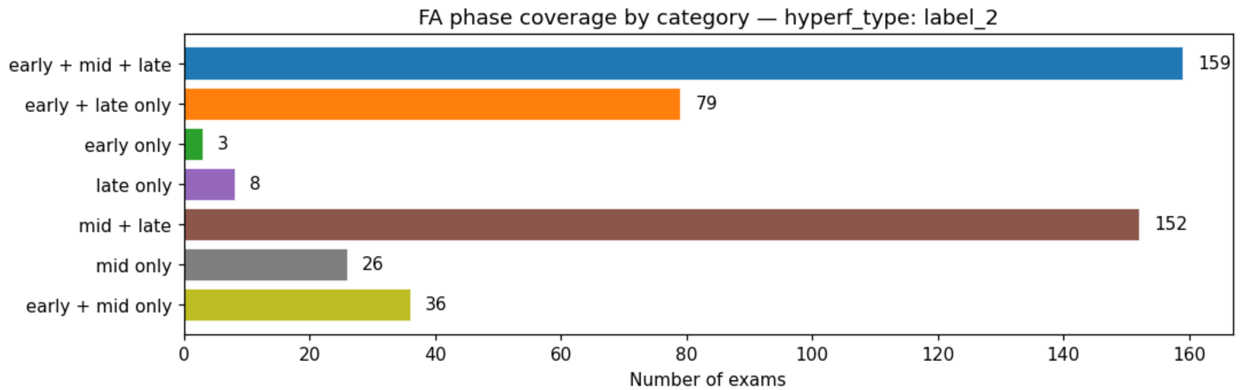
```python
# Collect stamped FA times per (hyperf_type, exam)
by_type_exam: dict[str, dict[str, list[float]]] = defaultdict(lambda:
defaultdict(list))
missing_label_exams: set[str] = set()

for r in rows:
    t = fa_seconds(r)
    if t is None:
        continue

    ex = str(r.get("exam") or "")
    if not ex:
        continue

    lbl = exam_to_hyperf.get(ex)
    if lbl is None:
        missing_label_exams.add(ex)
        continue

    htype = hyperf_name(lbl)
    by_type_exam[htype][ex].append(t)


# Count phase categories per hyperf_type
counts_by_type: dict[str, dict[str, int]] = {}
for htype, exam_map in by_type_exam.items():
    cat_counts_ht = defaultdict(int)
    for ex, ts in exam_map.items():
        ts_arr = np.asarray(ts, dtype=np.float64)
        cat_counts_ht[exam_phase_category(*phase_presence(ts_arr))] +=
1
    counts_by_type[htype] = cat_counts_ht

print(f"Mapped exams with stamped rows: {sum(len(v) for v in
by_type_exam.values()):,}")
if missing_label_exams:
    print(f"Stamped exams without HyperF label:
{len(missing_label_exams):,} (excluded)")


# Plot one chart per hyperf_type
types_sorted = sorted(counts_by_type.keys())
if not types_sorted:
    print("No labeled stamped rows found for per-hyperf_type phase
coverage.")
else:
    fig, axes = plt.subplots(
        nrows=len(types_sorted),
        ncols=1,
```

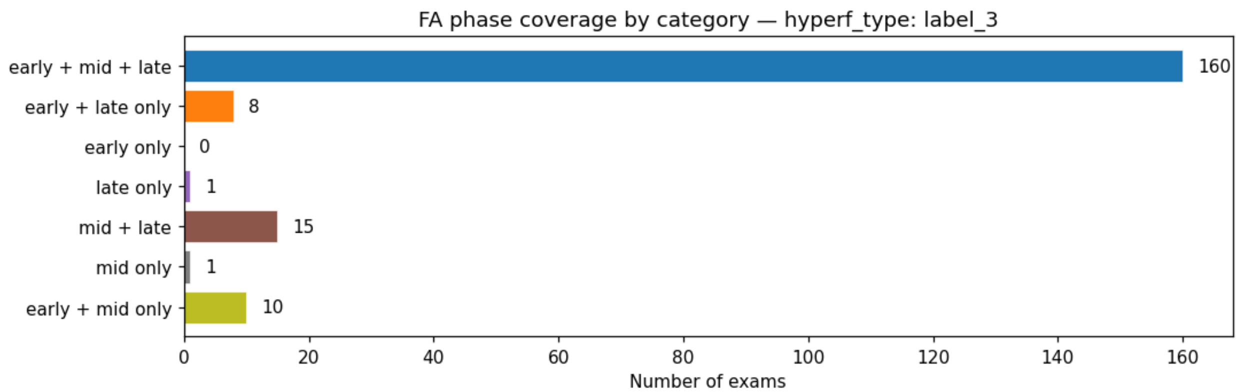
```
        figsize=(10, 3.2 * len(types_sorted)),
        squeeze=False,
    )

    colors = plt.cm.tab10(np.linspace(0, 0.85, len(order)))

    for i, htype in enumerate(types_sorted):
        ax = axes[i, 0]
        cat_counts_ht = counts_by_type[htype]
        counts_plot = [cat_counts_ht[k] for k in order]

        bars = ax.barh(order, counts_plot, color=colors,
edgecolor="white", linewidth=0.5)
        ax.invert_yaxis()
        ax.set_xlabel("Number of exams")
        ax.set_title(f"FA phase coverage by category — hyperf_type:
{htype}")

        xmax = max(counts_plot) if counts_plot else 1
        for b, c in zip(bars, counts_plot):
            ax.text(
                b.get_width() + xmax * 0.015,
                b.get_y() + b.get_height() / 2,
                f"{c:,}",
                va="center",
                fontsize=10,
            )

    fig.tight_layout()
    show_fig(fig)
```

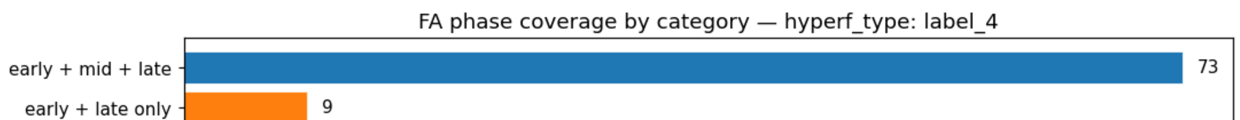
```
Mapped exams with stamped rows: 1,877
Stamped exams without HyperF label: 40 (excluded)
```


FA phase coverage by category — hyperf_type: label_0
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
601
49
1
1
59
4
23
0
100
200
300
400
500
600
Number of exams


FA phase coverage by category — hyperf_type: label_1
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
195
27
3
3
110
10
9
0
25
50
75
100
125
150
175
200
Number of exams


FA phase coverage by category — hyperf_type: label_2
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
159
79
3
8
152
26
36
0
20
40
60
80
100
120
140
160
Number of exams


FA phase coverage by category — hyperf_type: label_3
early + mid + late
early + late only
early only
late only
mid + late
mid only
early + mid only
160
8
0
1
15
1
10
0
20
40
60
80
100
120
140
160
Number of exams


FA phase coverage by category — hyperf_type: label_4
early + mid + late
early + late only
73
9

```
# --- Phase buckets (data-driven boundaries from separability search)
---
BOUND_EARLY_MID = 103.0
BOUND_MID_LATE = 518.0

exam_stamped_times: dict[str, np.ndarray] = {}
for r in rows:
    t = fa_seconds(r)
    if t is None:
        continue
    ex = str(r.get("exam") or "")
    exam_stamped_times.setdefault(ex, []).append(t)

for ex in exam_stamped_times:
    exam_stamped_times[ex] = np.asarray(exam_stamped_times[ex],
dtype=np.float64)


def phase_presence(sec: np.ndarray) -> tuple[bool, bool, bool]:
    """(has_early, has_mid, has_late) for stamped FA seconds."""
    has_e = bool(np.any(sec < BOUND_EARLY_MID))
    has_m = bool(np.any((sec >= BOUND_EARLY_MID) & (sec <
BOUND_MID_LATE)))
    has_l = bool(np.any(sec >= BOUND_MID_LATE))
    return has_e, has_m, has_l


def exam_phase_category(E: bool, M: bool, L: bool) -> str:
    if E and M and L:
        return "early + mid + late"
    if E and not M and L:
        return "early + late only"
    if E and not M and not L:
        return "early only"
    if not E and not M and L:
        return "late only"
    if not E and M and L:
        return "mid + late"
    if not E and M and not L:
        return "mid only"
    if E and M and not L:
        return "early + mid only"
    return "none"


cat_counts: dict[str, int] = defaultdict(int)
for ex, ts in exam_stamped_times.items():
    cat_counts[exam_phase_category(*phase_presence(ts))] += 1

all_exams_in_rows = {str(r.get("exam")) for r in rows if
r.get("exam")}
```

```
exams_no_stamp = all_exams_in_rows - set(exam_stamped_times)
n_no_stamp = len(exams_no_stamp)

print(
    f"Bounds: Early [0, {BOUND_EARLY_MID}), Mid [{BOUND_EARLY_MID},
{BOUND_MID_LATE}), Late [{BOUND_MID_LATE}, ∞) seconds\n"
)
print(f"Exams with ≥1 stamped FA frame: {len(exam_stamped_times):,}")
print(f"Exams with no stamped FA frame (skipped below):
{n_no_stamp:,}\n")

order = [
    "early + mid + late",
    "early + late only",
    "early only",
    "late only",
    "mid + late",
    "mid only",
    "early + mid only",
]
for label in order:
    print(f"  {label}: {cat_counts[label]:,}")
rest = sum(cat_counts[k] for k in cat_counts if k not in order and k !
= "none")
if cat_counts["none"] or rest:
    print(f"  other/none: {cat_counts['none'] + rest:,}")
```

```
Bounds: Early [0, 103.0), Mid [103.0, 518.0), Late [518.0, ∞) seconds

Exams with ≥1 stamped FA frame: 1,917
Exams with no stamped FA frame (skipped below): 4

  early + mid + late: 1,210
  early + late only: 173
  early only: 8
  late only: 16
  mid + late: 368
  mid only: 52
  early + mid only: 90
```

```
# Bar chart (exam counts; same category order as printed above)
counts_plot = [cat_counts[k] for k in order]

fig, ax = plt.subplots(figsize=(10, 6))
colors = plt.cm.tab10(np.linspace(0, 0.85, len(order)))
bars = ax.barh(order, counts_plot, color=colors, edgecolor="white",
linewidth=0.5)
ax.set_xlabel("Number of exams")
ax.set_title(
    "Exams by FA phase coverage (≥1 stamped frame per bin)\n"
```

```
    f"bounds early/mid {BOUND_EARLY_MID}s · mid/late
{BOUND_MID_LATE}s"
)
ax.invert_yaxis()
xmax = max(counts_plot) if counts_plot else 1
for b, c in zip(bars, counts_plot):
    ax.text(b.get_width() + xmax * 0.015, b.get_y() + b.get_height() /
2, f"{c:,}", va="center", fontsize=10)
fig.tight_layout()
show_fig(fig)
```